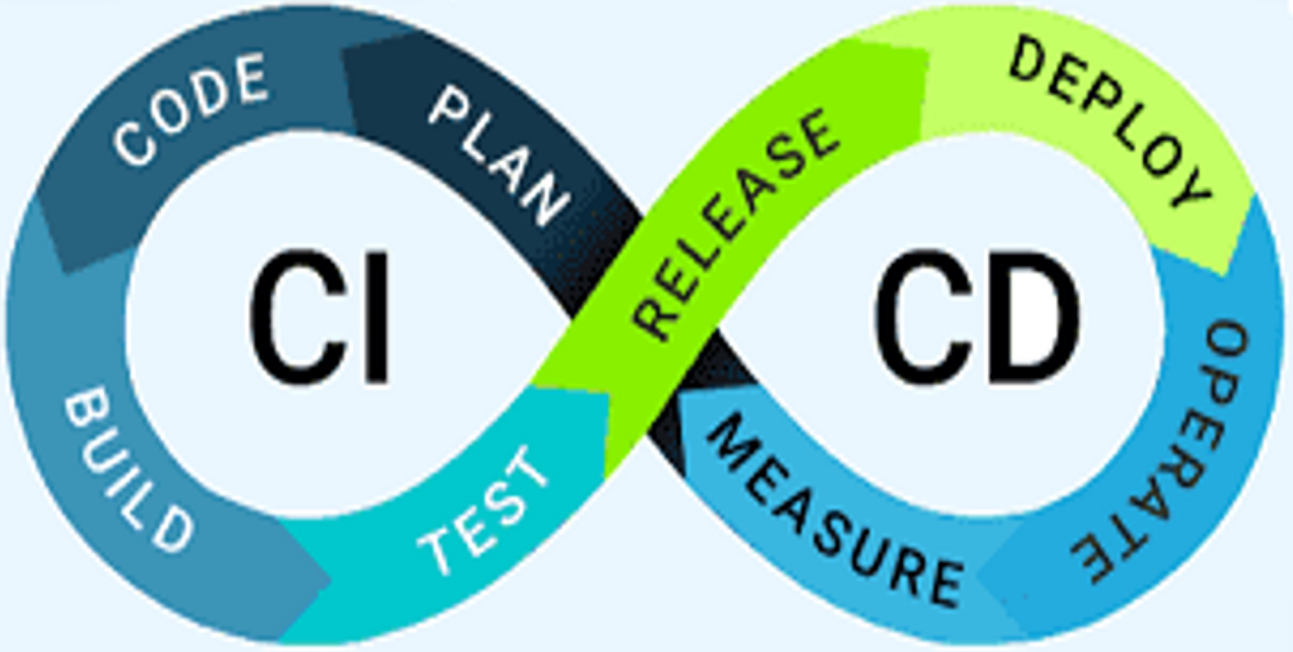
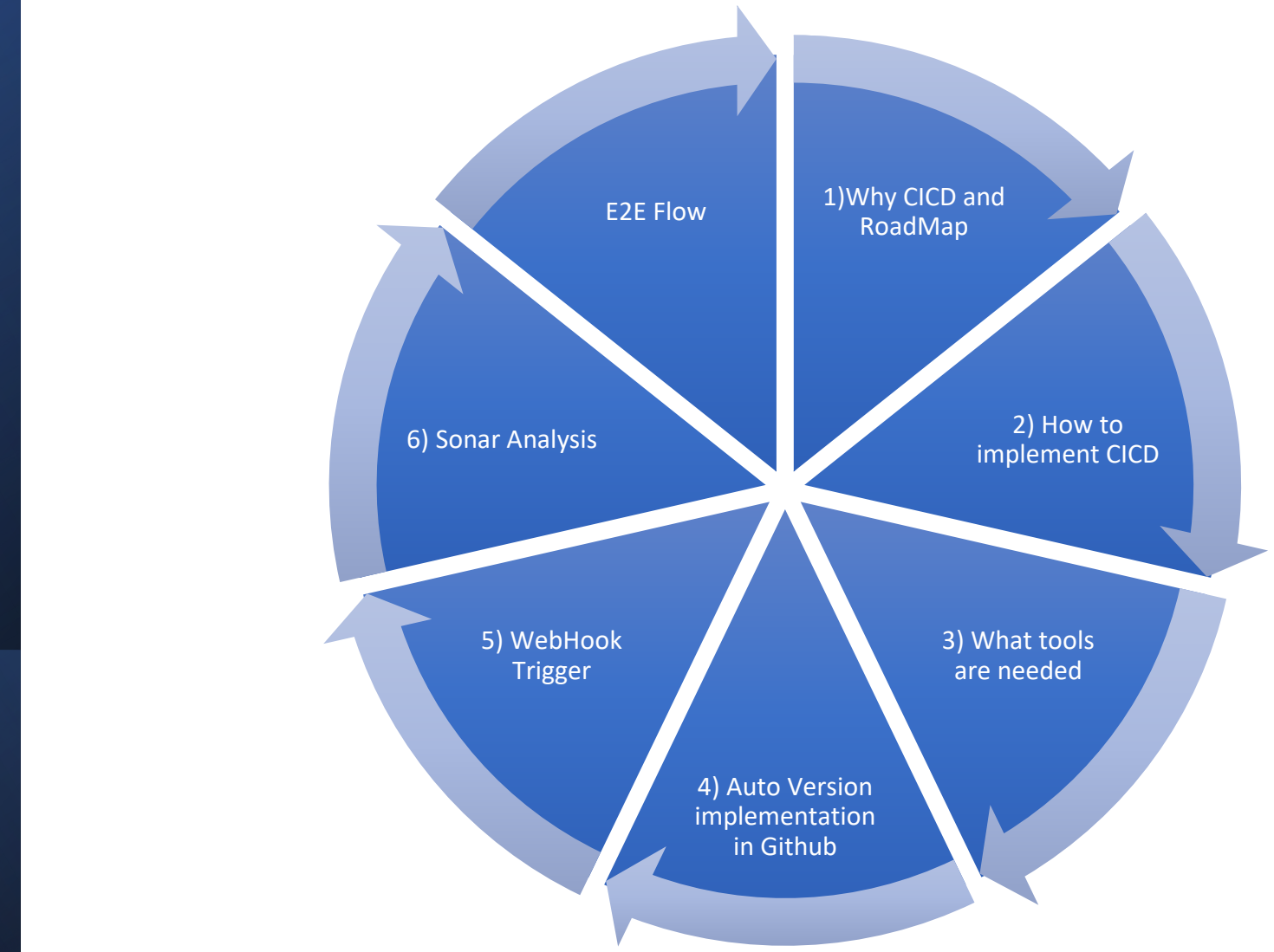
CICD End To End Flow

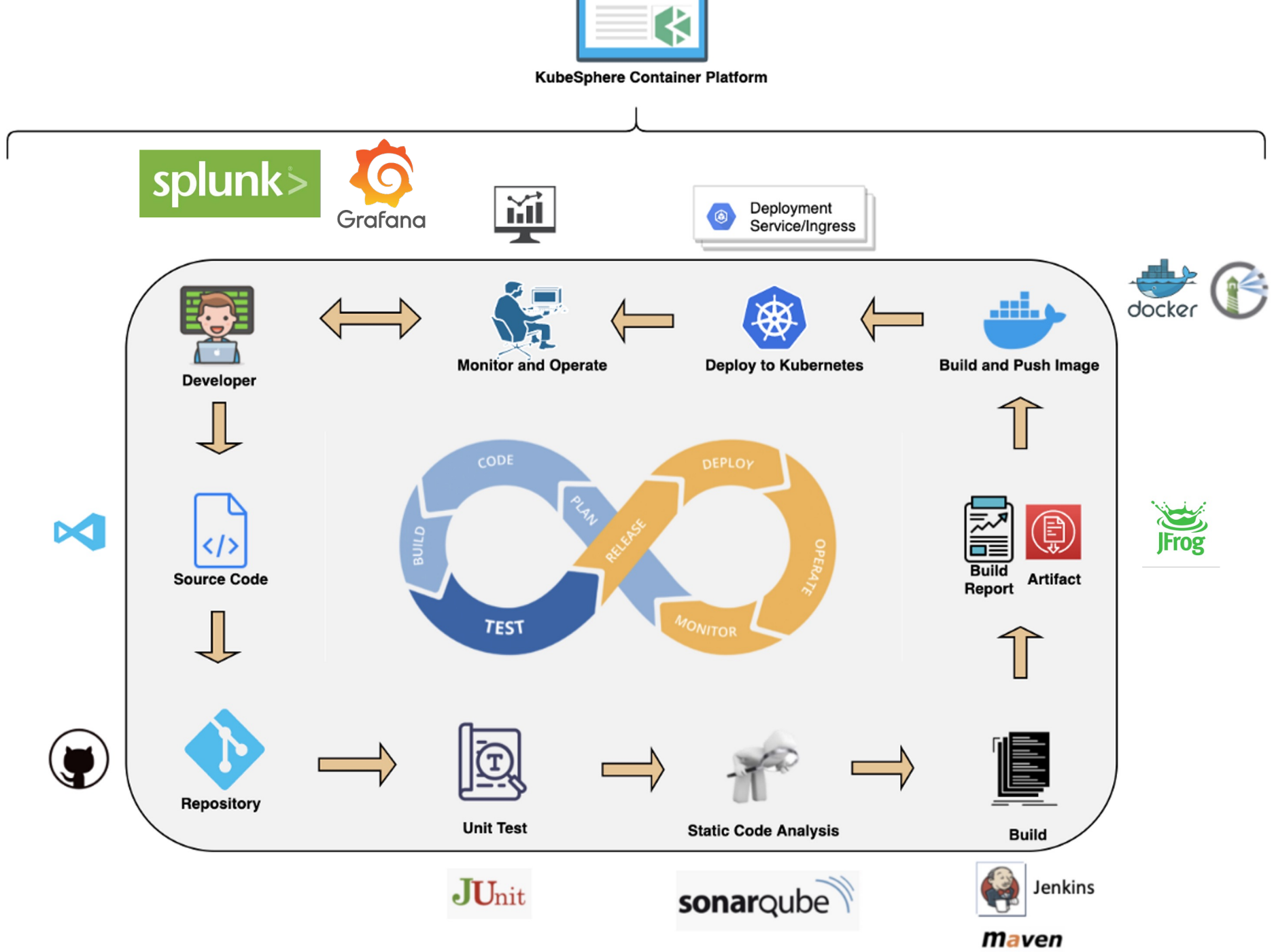
DevOps - CI/CD



By Praveen Singampalli

CICD discussion:




KubeSphere Container Platform
splunk>
Grafana
Deployment
Service/Ingress
docker
Developer
Monitor and Operate
Deploy to Kubernetes
Build and Push Image
Source Code
CODE
PLAN
BUILD
RELEASE
TEST
DEPLOY
OPERATE
MONITOR
Build
Report
Artifact
JFrog
Repository
Unit Test
Static Code Analysis
Build
JUnit
sonarqube
Jenkins
Maven

Different types of Jenkins Pipeline

- ***Declarative Pipeline Syntax***

The declarative syntax is a new feature that uses code for the pipeline. It provides a limited pre-defined structure. Thereby, it offers an easy & simple continuous delivery pipeline. Moreover, it uses a *pipeline block*.

- ***Scripted Pipeline Syntax***

Unlike declarative syntax, the *scripted pipeline syntax* is the old traditional way to write the *Jenkinsfile* on Jenkins web UI. Moreover, it strictly follows the groovy syntax and helps to develop a complex pipeline as code.

```
pipeline {
          agent any
          stages {
                    stage('One') {
                    steps {
                        echo 'Hi, welcome to pipeline demo...'
                    }
                    }
                    stage('Two') {
                    steps {
                       echo('Sample testing of Stage 2')
                    }
                    }
                    stage('Three') {

                    steps {
                          echo 'Thanks for using Jenkins Pipeline'
                    }
                    }
               }
}
```

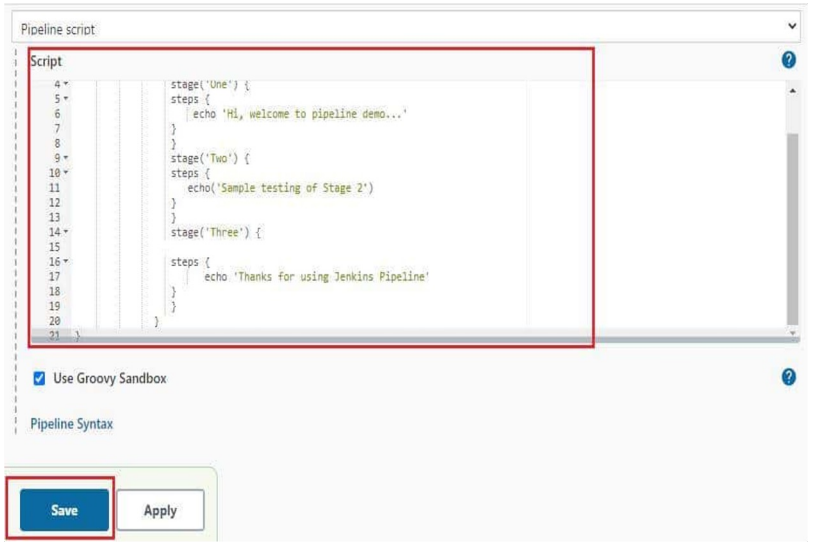
Declarative Pipeline

```
pipeline {
  agent {
    // executes on an executor with the label 'some-label' or 'docker'
    label "some-label || docker"
  }

  stages {
    stage("foo") {
      steps {
        // variable assignment (other than environment variables) can only
        // complex global variables (with properties or methods) can only
        // env variables can also be set within a script block
        script {
          foo = docker.image('ubuntu')
          env.bar = "${foo.imageName()}"
          echo "foo: ${foo.imageName()}"
        }
      }
    }
    stage("bar") {
      steps{
        echo "bar: ${env.bar}"
        echo "foo: ${foo.imageName()}"
      }
    }
  }
}
```

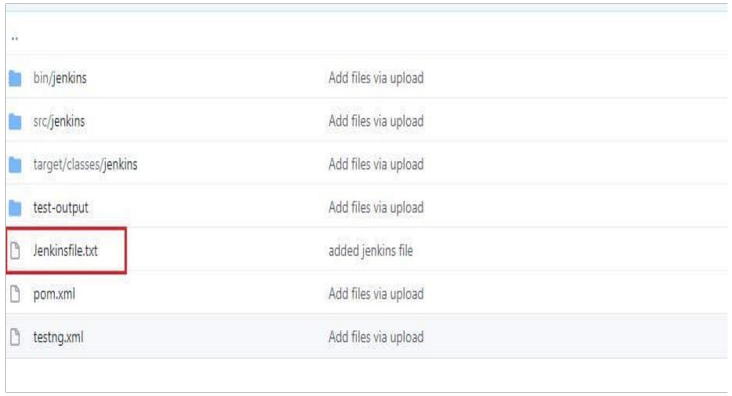
Scripted Pipeline

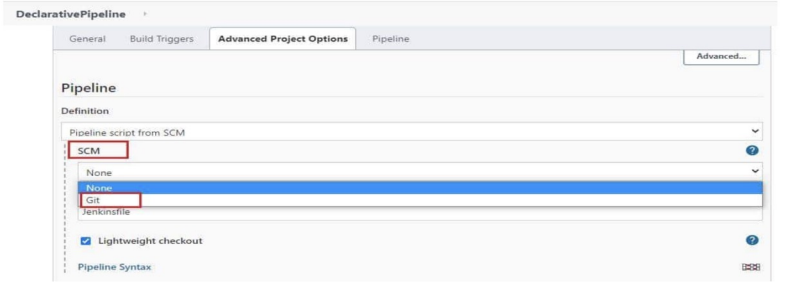
```
node {

  git url: 'https://github.com/jfrogdev/project-examples.git'

  // Get Artifactory server instance, defined in the Artifactory Plugi
  def server = Artifactory.server "SERVER_ID"

  // Read the upload spec and upload files to Artifactory.
  def downloadSpec =
       '''{
       "files": [
         {
            "pattern": "libs-snapshot-local/*.zip",
            "target": "dependencies/",
            "props": "p1=v1;p2=v2"
         }
       ]
   }'''

  def buildInfo1 = server.download spec: downloadSpec

  // Read the upload spec which was downloaded from github.
  def uploadSpec =
     '''{
     "files": [
       {
          "pattern": "resources/Kermit.*",
          "target": "libs-snapshot-local",
          "props": "p1=v1;p2=v2"
       },
       {
          "pattern": "resources/Frogger.*",
          "target": "libs-snapshot-local"
       }
      ]
    }'''
```

Scripted Pipeline

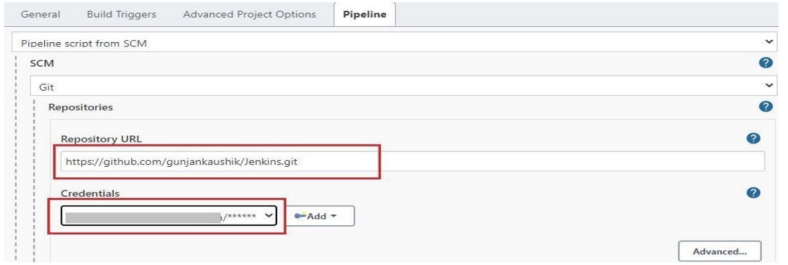


Declarative Pipeline(Jenkinsfile add to git repo)

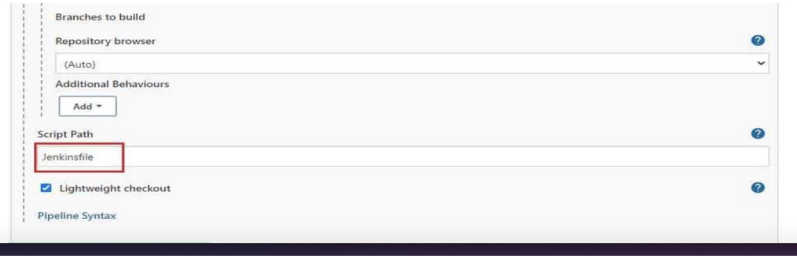




3. Now, you will get an option to input your *Repository URL* and *credentials*.



4. Next, you may set the ***branch*** or let it be blank for any branch. In the script path, you need to write the ***Jenkinsfile*** name that exists in your repository. Click on ***Save***, and there you go, your declarative pipeline is ready for use.



JenkinsFile Data

Key concepts of Jenkinsfile

- ***Pipeline*** *- A pipeline is a set of instructions that includes the processes of continuous delivery. For example, creating an application, testing it, and deploying the same. Moreover, it is a critical element in declarative pipeline syntax, which is a collection of all stages in a Jenkinsfile. We declare different stages and steps in this block.*
- pipeline{ } ***Node*** *- A node is a key element in scripted pipeline syntax. Moreover, it acts as a machine in Jenkins that executes the Pipeline.*
- node{ } ***Stage*** *- A stage consists of a set of processes that the Pipeline executes. Additionally, the tasks are divided in each stage, implying that there can be multiple stages within a Pipeline. The below snippet shows the different stages that one can define in a Pipeline.*
- ***Steps*** *- A step in Jenkins defines what we have to do at a particular step in the process. There can be a series of steps within the stage. Moreover, whatever step we define in a stage would be executed within it.*
- ***Agent*** *- An agent is a directive that enables the users to execute multiple projects within the same Jenkins instance by distributing the load. Moreover, we assign an executor to the build through an agent. You can either use a single agent for the entire pipeline or use a distinct agent for the different stages of the pipeline. Subsequently, some of the parameters used with agents are -*
  - ***Any****- Any of the available agents execute the pipeline.*
  - ***None****- It is used at the pipeline root and implies no global agent, but each stage must specify its own agent.*
  - ***Label****- The labeled agent is used to execute the pipeline or the specific stage.*
  - ***Docker****- One can use the Docker images as the execution environment & specifying the agent as docker.*

Merge Request from dev to master Branch

Child Pipeline -

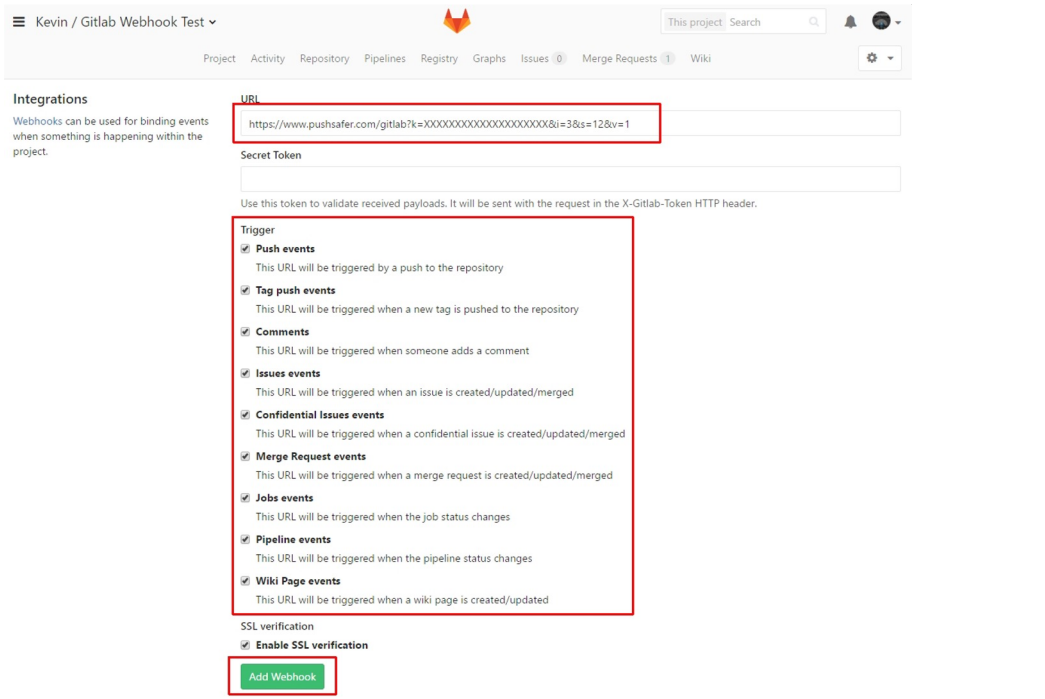
DevTeam Code Commit To GitHub Triggers Child Pipeline (WebHook) Sonar/Blackduck/Fortify(Stage 1 - In Parallel) Creates the Merge Request(Stage 2 – Sends Email to Approver)

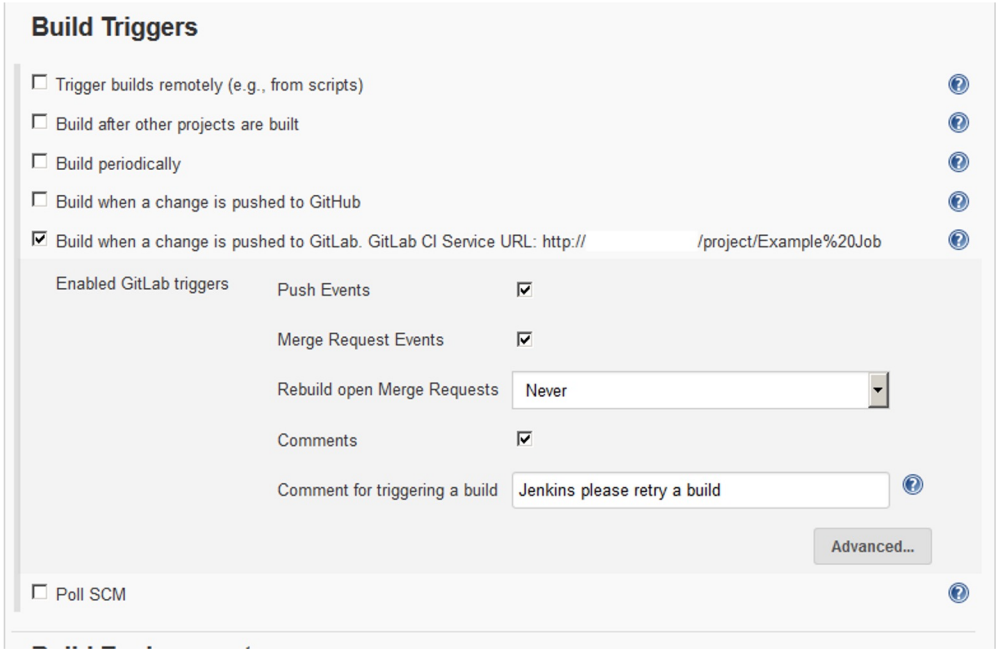
Master Pipeline -

Merge request gets approved Master Pipeline Trigger(WebHook) Sonar/Blackduck/Fortify(Stage 1 - In Parallel) Build Code (Maven) Artifact Push (Jfrog) Pull Artifact(Ansible) Deployment to QA(OnPrem/Cloud) Performance testing/Integration testing Deployment to Syage Send Notification via Email

Webhook Setup

- In your project or group, on the left sidebar, select Settings > Webhooks.
- In URL, enter the URL of the webhook endpoint.
- In Secret token, enter the secret token to validate payloads.
- In the Trigger section, select the events to trigger the webhook

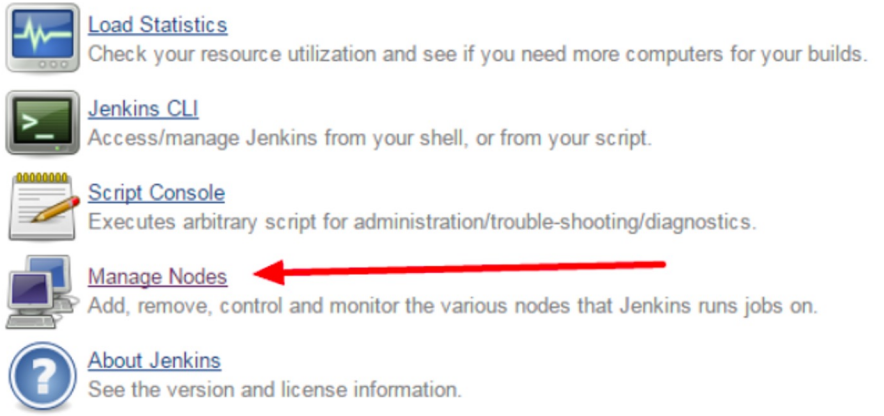




Jenkins Master slave Config

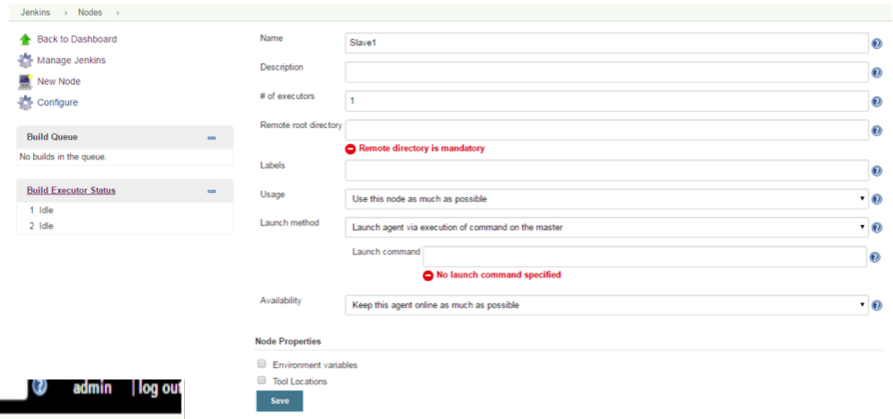
1

2

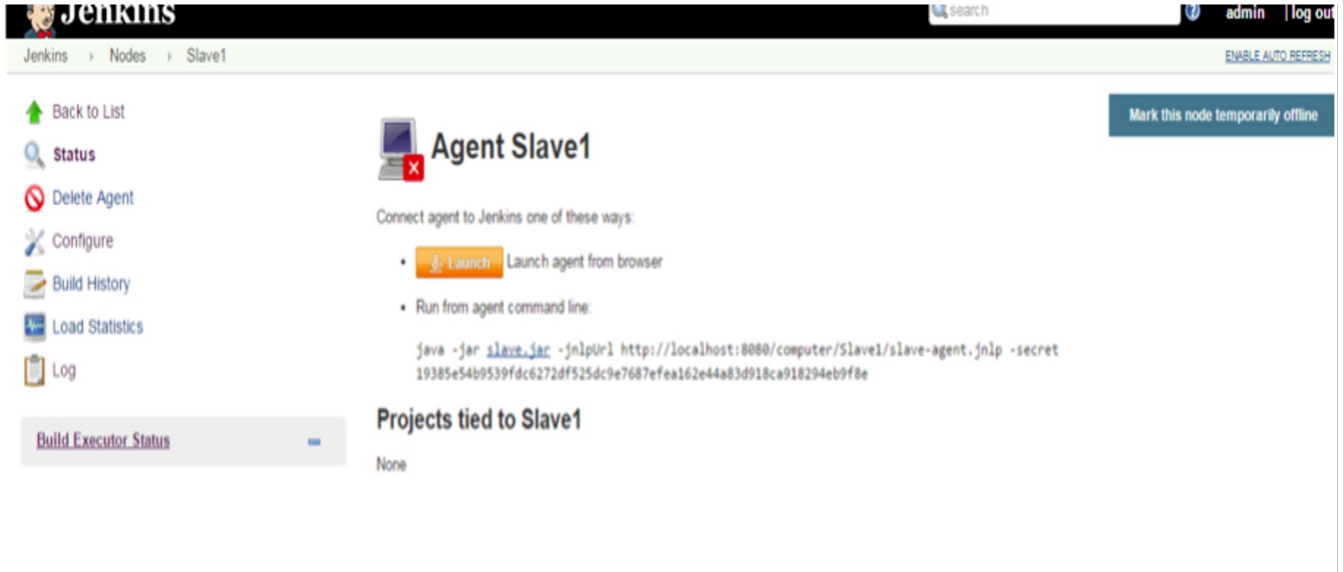


3

4



5



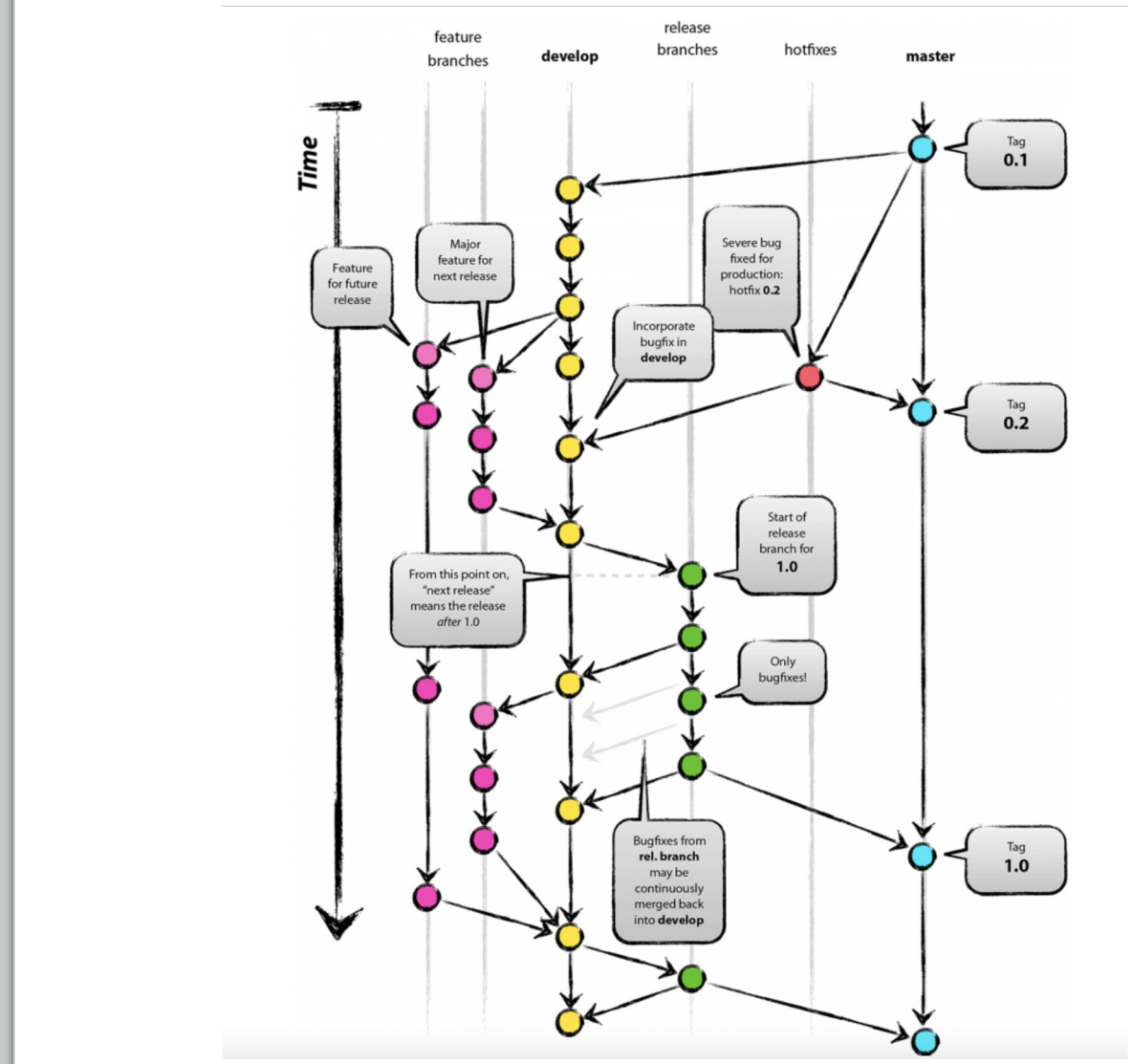
Trigger a pipeline from JenkinsFile

```
pipeline {
    agent
    {
        node {
                label 'master'
                customWorkspace "${env.JobPath}"
             }
    }

    stages
    {
        stage('Start') {
            steps {
                sh 'ls'
            }
        }

        stage ('Invoke_pipeline') {
            steps {
                build job: 'pipeline1', parameters: [
                string(name: 'param1', value: "value1")
                ]
            }
        }

        stage('End') {
            steps {
                sh 'ls'
            }
        }
    }
}
```

```
pipeline {
    agent {
        docker {
            image 'node:6-alpine'
            args '-p 3000:3000 -p 5000:5000'
        }
    }
    environment {
        CI = 'true'
    }
    stages {
        stage('Build') {
            steps {
                sh 'npm install'
            }
        }
        stage('Test') {
            steps {
                sh './jenkins/scripts/test.sh'
            }
        }
        stage('Deliver for development') {
            when {
                branch 'development'
            }
            steps {
                sh './jenkins/scripts/deliver-for-development.sh'
                input message: 'Finished using the web site? (Click "Proceed" to continue)
                sh './jenkins/scripts/kill.sh'
            }
        }
        stage('Deploy for production') {
            when {
                branch 'production'
            }
            steps {
                sh './jenkins/scripts/deploy-for-production.sh'
                input message: 'Finished using the web site? (Click "Proceed" to continue)
                sh './jenkins/scripts/kill.sh'
            }
        }
    }
}
```

GitLab Branching Strategy

- **Master Branch** – This branch mainly deals with the prod deployment(Prod Pipeline)
- **Hotfix Branch** – This branch deals with any issues related to prod fixes (Bug/Feedback from customer)(Hotfix Pipeline)
- **Release Branch** (Release Pipeline) – The code from this branch is deployed in higher environments(QA/STAGE) except prod
- **Develop Branch** (Dev Pipeline)– Called as Dev branch and it deals with the changes done by developers for their own branch
- **Feature Branch –** Deals with the new features taken up for the sprint(We usually use to merge the feature to release branch and then deploy)



TOOLS INSTALLATION

SOFTWARES DOWNLOAD

1. **Visual Studio Code [MAC AND WINDOWS]**

https://code.visualstudio.com/download

2. Python Install. —> python3 --version

- https://www.python.org/downloads/

3. Install brew if it is not there in MAC [ONLY FOR MAC USERS]

- open the terminal and copy below command

STEP 1 - /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

STEP 2 - vim .zshrc

STEP 3 - now enter i to start edit mode and write

- export PATH=/opt/homebrew/bin:$PATH
- STEP 4 - source ~/.zshrc

4. **Terraform**. https://www.terraform.io/downloads -> terraform -v

- ONLY IF TERRAFORM GIVES ERROR FOR MAC
- brew install kreuzwerker/taps/m1-terraform-provider-helper
- m1-terraform-provider-helper activate # (In case you have not activated the helper)
- m1-terraform-provider-helper install hashicorp/template -v v2.2.0 # Install and compile

5. AWS ACESS

- https://aws.amazon.com/console/
- Sign In for FREE console
- It will ask for credit card details
- Keep the sign in details safe

6. Install aws cli - https://docs.aws.amazon.com/cli/v1/userguide/install-macos.html

- If PYTHON error comes on MAC then
- sudo /usr/local/bin/python3.11 awscli-bundle/install -i /usr/local/aws -b /usr/local/bin/aws

7. **AWS CONFIGURE**

- CREATE THE USER IN AWS -> SAVE THE ACCESS KEY AND SECRET KEY

9. GIT -> https://git-scm.com/download/win

10. Install Docker

- https://docs.docker.com/desktop/install/mac-install/ -> MAC
- https://docs.docker.com/desktop/install/windows-install/ -> WINDOWS

11. JAVA INSTALL :

- https://jdk.java.net/18/
- https://www.digitalocean.com/community/tutorials/install-maven-mac-os

- sudo mv jdk-18.0.2.1.jdk /Library/Java/JavaVirtualMachines/
- JAVA_HOME="/Library/Java/JavaVirtualMachines/jdk-18.0.2.1.jdk/Contents/Home"
- PATH="${JAVA_HOME}/bin:${PATH}"
- export PATH
- java -version

12. MAVEN INSTALL:

MAC - brew install maven

WINDOWS - https://maven.apache.org/download.cgi

13. MINIKUBE INSTALL:

- https://minikube.sigs.k8s.io/docs/start/

14. NODE/NPM INSTALL:

INSTALL NODE - https://nodejs.org/en/download/